Exercise 3

Identify the exact assembly instructions that correspond to each of the statements in readsect()

readsect()	**asm**
`waitdisk()`	`0x7cf4: call 0x7cda`

`outb(0x1F2, 1);`	`0x7cf9: mov $0x1f2,%edx`
	`0x7cfe: mov $0x1,%al`
	`0x7d00: out %al,(%dx)`

`outb(0x1F3, offset);`	`0x7d01: movzbl %bl,%eax`
	`0x7d04: mov $0xf3,%dl`
	`0x7d06: out %al,(%dx)`

`outb(0x1F4, offset >> 8);`	`0x7d07: movzbl %bh,%eax`
	`0x7d0a: mov $0xf4,%dl`
	`0x7d0c: out %al,(%dx)`

`outb(0x1F5, offset >> 16);`	`0x7d0d: mov %ebx,%eax`
	`0x7d0f: mov $0xf5,%dl`
	`0x7d11: shr $0x10,%eax`
	`0x7d14: movzbl %al,%eax`
	`0x7d17: out %al,(%dx)`

`outb(0x1F6, (offset >> 24) | 0xE0);`	`0x7d18: shr $0x18,%ebx`
	`0x7d1b: mov $0xf6,%dl`
	`0x7d1d: mov %bl,%al`
	`0x7d1f: or $0xffffffe0,%eax`
	`0x7d22: out %al,(%dx)`

`outb(0x1F7, 0x20);`	`0x7d23: mov $0x20,%al`
	`0x7d25: mov $0xf7,%dl`
	`0x7d27: out %al,(%dx)`

`waitdisk();`	`0x7d28: call 0x7cda`

`insl(0x1F0, dst, SECTSIZE/4);`	`0x7d2d: mov 0x8(%ebp),%edi`
	`0x7d30: mov $0x80,%ecx`
	`0x7d35: mov $0x1f0,%edx`
	`0x7d3a: cld`
	`0x7d3b: repnz insl (%dx),%es:(%edi)`

Identify the begin and end of the for loop that reads the remaining sectors of the kernel from the disk

Beginning: 0x7db6, end: 0x7dce

```
0x7db6:  cmp    %esi,%ebx
0x7db8:  jae    0x7dd0
```

```
0x7dba:  pushl  0x8(%ebx)
0x7dbd:  add    $0x38,%ebx
0x7dc0:  pushl  -0x10(%ebx)
0x7dc3:  pushl  -0x20(%ebx)
0x7dc6:  call   0x7d41
0x7dcb:  add    $0xc,%esp
0x7dce:  jmp    0x7db6
```

Find out what code will run when the loop is finished, set a breakpoint there, and continue to that breakpoint

1. **At what point does the processor start executing 32-bit code? What exactly causes the switch from 16- to 32-bit mode?**
    a. Right before the call into bootmain, it switches to 32 bit. Setting the protected mode flag and then doing a long jump causes the switch.

2. **What is the last instruction of the boot loader executed, and what is the first instruction of the kernel it just loaded?**
    a. Last (two) instructions of bootloader:
    ```
    0x7dd5:  mov    %eax,%ebx
    0x7dd7:  call   *0x10018
    ```
    b. First of kernel:
    ```
    movl $multiboot_info, %eax
    ```

3. **How does the boot loader decide how many sectors it must read in order to fetch the entire kernel from disk? Where does it find this information?**
    a. In line 54 of boot/main.c, it checks whether the program has written to 0x10000, and loads in a segment when it hasn't. It gets all the information from the ELF header.

Exercise 5

Identify the first instruction that would "break" or otherwise do the wrong thing if you were to get the boot loader's link address wrong.

1. The first thing to break will be the transition into protected mode, because this depends on the link address being right for the GDT segment translation. The first instruction that won't work is `ljmp    $PROT_MODE_CSEG, $protcseg`

Exercise 9

Determine where the kernel initializes its stack, and exactly where in memory its stack is located. How does the kernel reserve space for its stack? And at which "end" of this reserved area is the stack pointer initialized to point to?

1. The kernel's stack is located at 0x8004000000. The kernel reserves space by clearing the frame pointer by setting it to 0 and then setting the stack pointer, before it calls the i386init function. The "top" of this reserved area is where the stack pointer starts, and the stack grows downwards.